

Code quality: branches, closed vocabularies and literals

2026-10-17, branch 20261017-codequality, at 638b5bdc. Analysis and plan only: nothing in the code is changed by this document.

1. Scope and method

Scope. The production C++ of eacmplib, qtgui, tools and bindings: 259 files, 143,949 lines. The two frozen migration ledgers (tools/migrate_landscapes.cpp, tools/migrate_preferences.cpp) are left out, as the audits leave them out. mechanical_tests is counted separately where it matters.

Method. Three sources:

- A scanner, kept in the session scratchpad and not committed. It is a small lexer that blanks comments and literal bodies, with bracket matching on top. It counts switches, if chains, numeric literals and string literals. It is not a compiler, so its counts are close but not exact. Every example cited below was checked by reading the code.
- A syntax-only compile of the 133 production translation units with clang. It used the build tree's compile commands plus -Wswitch-enum -Wcovered-switch-default -Wswitch-default -Wimplicit-fallthrough.
- Reading the rules already in force:
 - EACMP-documentationincode.md §7.1 and §9;
 - EACMP-software-architecture.md, *Invariants and Design Rules*: the default: rule and *Derived coverage*;
 - tools/audit_comments.py.

2. Summary

#	Finding	Size
F1	The [else] audit rule misses the code base's usual forms: if (...) { // comment, } else if, and one-line bodies. Its zero findings say almost nothing.	3,005 fall-through ifs and else if tails in production lacking the marker where it belongs; 7,797 unmarked guard clauses
F2	Two closed grammar vocabularies, rule action and hideRelations, are read by an if/else if chain with no final else. A misspelt value is accepted silently and the struct's default value stays.	2 defects (grammar_load.cpp:776, :798)
F3	Closed vocabularies are represented five different ways: an enum with a refusing converter, an enum with a fallback converter, an inline chain, raw compares into bool/int, or a string kept until use. Each vocabulary is spelled out again in several places.	19 / 6 / 2 / 4 / 1 vocabularies; 41 inline string-dispatch chains; log levels parsed by 5 separate converters
F4	Label placement shape is an enumerator that one function cannot take. Its pairing rule is kept by hand in three places. The landscape loader and the schemas do not check the vocabulary.	label_geometry.cpp:18, case study in §3
F5	After an exhaustive enum switch whose cases all return, the trailing line is a plausible return value. The rule requires EACMP_BREAKING_RECOVERABLE_INVARIANT.	42 of 49 sites
F6	-Wswitch is the agreed guard, but warnings are not errors. Partial switches with a default: hide new enumerators. Three defaults are dead.	24 internal-enum sites under -Wswitch-enum; 3 dead defaults

#	Finding	Size
F7	Ints are cast to project enums without a check: combo indices, Qt item data, compiled-cache numbers.	53 casts
F8	Inline numeric literals outside named definitions.	1,783 (qtgui 1,117, eacmplib 524, tools 141)
F9	String literals used as protocol: JSON keys, compared tokens, file suffixes, and one display text used as a sentinel.	1,707 key reads (539 keys); 597 compares (340 values)

The rules that F1, F5 and F6 need are already written down. The audit does not enforce them, and neither does the build. F2 to F4 come from one missing pattern: a single descriptor per closed vocabulary. The project already uses that pattern elsewhere (`scheme_tokens.h`, `referral_chains.h`, the field registries).

3. Case study: `EACMP_HPlacement::Shape` (`label_geometry.cpp:18`)

What shape is. It is a valid value for a grammar's `labelPlacementH`/`labelPlacementV` (grammar reference, style-scheme table). It is explained after that table ("The value shape means the label area is the shape's own `labelArea rect ...`") and in *Shape-anchored label areas*. So it is documented. The "should never reach" in the code is about this one function, not about the grammar value.

Can line 37 be reached?

- `EACMP_resolveViewLabelArea` (`view_geometry.cpp:92`) routes either axis being `Shape` to the shape-area branch. The call at line 141 is reached only when neither axis is `Shape`.
- The auto-placement row orders (`label_geometry.cpp:89`, `view_gui_internal.h:1458`) never contain `Shape`.

So the case is unreachable unless a caller breaks the function's contract. That is a programming error, and the rule for it is `EACMP_BREAKING_RECOVERABLE_INVARIANT`. The code instead logs at Error and centres the label, with "fallback justified". The same holds for the vertical switch at line 45.

Root cause. The type admits a value that the function cannot take. `EACMP_HPlacement` mixes six box placements with a whole-rect mode. `types.h:214` states the invariant that the mode is "always set on BOTH axes together". Only some paths enforce it, each by hand:

Path	Unknown string	Half pair (shape on one axis)
Grammar scheme rule (<code>grammar_load_schemes.cpp:98–104</code>)	stored as text, not refused	refused
Landscape item (<code>view_undo_state.cpp:241–242</code>)	stored as text, not refused	not checked
Properties panel (<code>view_window_props.cpp:1133, :1186</code>)	n/a	the other axis is paired by hand
Geometry (<code>view_geometry.cpp:69, :83, :92</code>)	<code>EACMP_hPlacementFromString</code> logs Error and returns h-center on every resolve	\\ \\ tolerates it
Schemas (<code>grammar.schema.json:342, appearanceschemes.schema.json:141, landscape.schema.json:593</code>)	"type": "string", no enum	no check

The vocabulary is also spelled out four times:

- `types.cpp:16–70`, once to string and once from string, per axis;
- the panel's combo items (`view_window_props.cpp:1102–1111` and the vertical copy);
- the grammar reference.

The combo maps its index onto the enum by position (`static_cast<int>(...) + 1, static_cast<EACMP_HPlacement>(placementI - 1)` at `view_window_props.cpp:1116, :1130, view_window.cpp:6038, :6050`). So the enum's order is a hidden contract with the combo.

Proposed fix.

1. Store each axis as a typed property (`EACMP_Property<EACMP_HPlacement>`), not a string.
2. Resolve to a type that cannot hold a half pair: *shape area* or *box (H, V)*, where the box enums have no `Shape`.
3. `EACMP_computeLabelAreaRect` takes the box enums only. Its switches become exhaustive with no impossible case, and the trailing line is the invariant.
4. Refuse an unknown value and a half pair at both loaders (GR-1), and in the setter.

5. Give the schemas the `enum`.

6. Build the combos from the vocabulary descriptor (§6) and store the `enum`, not the index.

The wire format stays the same: two strings, `shape` on both axes. No format bump is needed. Existing landscapes must be checked for half pairs and unknown values, and repaired by the migration tool if any exist.

4. switch

The rule in force (architecture, default: `rule`; documentation §7.1):

- over a closed internal `enum` with every enumerator handled, write `no default:`, and make the trailing line `EACMP_BREAKING_RECOVERABLE_INVARIANT`;
- over boundary input, write a `default:` that refuses;
- over a third-party `enum`, write a `default:` whose comment says “not interested”.

`goal.txt` asks for “always a default unless there is a good reason, with a comment”. The `-Wswitch` check is that good reason. What is missing is the comment, and enforcement.

Counts: 162 production switches.

Kind	With default:	Without
Internal enum (qualified labels)	22	112 (111 exhaustive, 1 not resolved by the scanner)
Unscoped enum or constants	10	7
Integer or character	5	0
Third-party enum (Qt, nlohmann)	6	0

S1. `-Wswitch` warns but does not fail. The compile contract (`CMakeLists.txt:33–39`) is `-Wall -Wextra -Wpedantic` on clang and `/W4` on MSVC, with no `-Werror`. A forgotten enumerator compiles with a warning.

The tree has four warnings today. They are outside this brief and reported only:

- `relation_reroute.cpp:1123, :1138, :1146: -Wmissing-field-initializers`;
- `undo.cpp:3669: a captured structured binding, a C++20 extension`.

S2. The line after an exhaustive switch whose cases all return. The scanner finds 49 sites. A few of them are switches whose cases break (for example `tree_browser.cpp:3322`), so the counts are approximate.

Trailing line	Sites	Examples
<code>EACMP_BREAKING_RECOVERABLE_INVARIANT</code> , as the rule requires	5	<code>logic.cpp:231</code> , <code>import.cpp:456</code>
Error log, then a plausible value	10	<code>types.cpp:27</code> (“h-center”), <code>grammar.cpp:516</code> , <code>:535</code> , <code>:776</code> , <code>view_gui.cpp:194</code>
A plausible value, commented “unreachable ... present only to satisfy the compiler”	about 30	<code>relation_routing.h:44</code> , <code>view_aspects.h:36</code> , <code>types.h:157</code> , <code>logging.cpp:212</code> , <code>grammar_compile.cpp:1111</code> (“?”)
A refusal value (<code>makeError</code> , <code>luaL_error</code>)	2	<code>landscape_variables.cpp:717</code> , <code>logic_lua.cpp:608</code>
Falls out of a void function	2	<code>view_window.cpp:4270</code> , <code>:4297</code>

S3. Exhaustive switches whose cases fall out (assign, then `break`). There are 63, such as `label_geometry.cpp:18`. They are sound only if the switched value is a real enumerator. Inside the program that holds, except where an `int` is cast to the `enum` (§6).

S4. Partial switches over internal enums with a `default:`. There are 22, all commented as `[default]` requires, for example `action_registry.cpp:68`, `tree_browser.cpp:1318`, `view_window.cpp:3609`. The `default:` takes every enumerator added later without a word from the compiler. `-Wswitch-enum` reports 28 sites:

- these 22;
- two over unscoped internal enums (`landscape_variables.cpp:426`, `undo.cpp:2408`);
- four over third-party enums (`logic.cpp:161`, `keys_overlay.cpp:246`, `script_player_actions.cpp:93`, `script_recorder.cpp:304`). The remedy is to name the other enumerators as grouped cases sharing one body, and drop the `default:`.

S5. Dead defaults. `undo.cpp:2543`, `:2713` and `:2772` have a `default:` in a switch that already covers every enumerator (`-Wcovered-switch-default`). They never run, and they switch `-Wswitch` off.

S6. Ints cast to project enums. There are 53 `static_cast`s:

- tree_browser.cpp: 26, mostly from item data (KindRole);
- the placement combos (§3);
- the compiled cache: grammar_compile.cpp:2980, :2993 write enums as ints, and :3290–:3310 read them back with jsonInt(..., 0);
- logic_lua.cpp:211, :1103, :1178, :1181; app_manager.cpp:3646, :3654; others.

The rule names “an int cast to an enum” as boundary input. Each cast needs a checked conversion, for example one that returns std::optional. After it, the switches downstream are internal again.

S7. Unscoped enums. There are 11, beside 176 enum class:

- undo.h:665, :700, :761, :781, :1308;
- validation.h:13, grammar.h:1071, constraint_expr.h:16, landscape_variables.h:110;
- view_gui.h:180, view_window.cpp:120.

They convert to int implicitly, and their enumerators leak into the enclosing scope.

5. if / else

The rule in force (§7.1, audit rule 5 [else]): an if has an else, or its opening comment explains why not and ends with the no 'else' marker.

Counts: 14,921 ifs form 14,360 chains in production. 675 chains end in a final else and 13,685 do not.

Chain without a final else	Marker in the opening comment	Marker on the closing brace	No marker
Guard: every branch ends in return/continue/break/throw	1,281	11	7,797 (2 of them else if tails)
Single if, falls through	1,567	12	2,799
if ... else if ... with no final else, falls through	24 (on the last branch)	18	171, plus 5 marked on another branch

Why the audit reports zero. I confirmed this by calling audit_file on a probe file:

- **H1.** Rule 5 checks only an if whose line ends in {, or whose { is on the next line (audit_comments.py:455–462). The house form if (...) { // comment ends in the comment, so it is never checked.
- **H2.** FLOW_KEYWORDS (audit_comments.py:176) matches only lines that begin with if or else if. A } else if (...) { line is never examined. Worse, the chain head counts the else if after it as its else. So a chain ending in else if always passes. This is goal.txt’s observation.
- **H3.** One-line and brace-less bodies are “not covered” (:461). Rule 2 [flow] has the same holes: 2,517 one-line ifs carry no comment at all.

What a missing else costs.

- grammar_load.cpp:776–789: the rule action chain ends in else if. An unknown action keeps EACMP_RuleAction::AddAsCoreToAllowedTable, the member’s default (grammar.h:992).
- :798–800: hideRelations does the same.
- In the same function, phase, binding and layout refuse unknown values through converters (:759, :803, :809).

Decision needed: guard clauses. A branch that ends in a jump has the rest of the block as its else. Requiring the marker on 7,797 guards adds a ritual comment and no information. Recommendation: count a jump-terminated branch as complete, say so in §7.1, and require the marker only where the false case falls through.

Work list once the audit is repaired. 3,005 production sites:

- 2,970 fall-through sites with no marker;
- 5 tails marked on another branch;
- 30 with the marker on the closing brace, which moves to the opening comment.

Another 252 are in mechanical_tests.

Files with the most open sites:

File	Sites
grammar_compile.cpp	269
view_window.cpp	197
grammar_load.cpp	149

File	Sites
view_window_props.cpp	142
import.cpp	135
app_manager.cpp	126
view_gui_mouse.cpp	126
tree_browser.cpp	105

The loaders, import and serialization hold 567 of them. That is where a missing `else` means input is accepted without a check, so they come first.

6. Closed vocabularies: enums or strings

goal.txt notes that some grammar keys become enums and others do not. For *closed* vocabularies (a field whose value is one of a fixed set) the code uses five representations:

Representation	Vocabularies	Unknown value
A. Enum, refusing converter <code>bool XFromString(const std::string&, X&)</code>	19: routing style, multiplicity use/form, attachment kind, shape scaling, icon anchor, instantiable, canvas role, nesting binding/layout, displayAsContainment, rule phase, derivation result/direction/position, label display, view override mode, node-type mode, import scan kind	refused by the caller
B. Enum, fallback converter <code>X XFromString(const std::string&)</code>	6: hPlacement, vPlacement, labelVAlign, labelHAlign, labelFlush (types.cpp), log level (logging.cpp:308, fallback parameter)	Error log and a default value. The landscape loader uses these (view_undo_state.cpp:233–242), so a bad value in a file is replaced, not refused.
C. Enum, mapped by an inline if chain	2: rule action, hideRelations (grammar_load.cpp:776, :798)	accepted silently (F2)
D. No enum: raw compares into <code>bool</code> , <code>int</code> or <code>bits</code>	4: defaultLabelDisplay → <code>bool</code> (grammar_load_appearances.cpp:73); stackAxis → <code>int</code> 0/1, -1 = none (:107, grammar.h:564, nesting_layout.h:136); rule log events → <code>bits</code> 1/2/4, 7 = all (grammar_internal.h:222); constraint category kept as a string (grammar_load_constraints.cpp:193, :224, :328)	refused, except events, where an unknown name gives 0
E. String kept in the model and parsed at use	label placement (§3)	Error log at every resolve

Each vocabulary is spelled out several times. Typically:

1. a from-string if chain;
2. a to-string switch;
3. the schema `enum`, where there is one;
4. the grammar reference;
5. sometimes a GUI combo or a test.

The scanner finds 41 string-dispatch chains of three or more compares of one variable (218 compares), for example grammar.cpp:683 (14 scalings) and grammar.cpp:641, :656. Log-level names have five parsers:

- logging.cpp:308, capitalised: "Debug";
- grammar_internal.h:211, lowercase;
- tools/compare_relation_tables.cpp:78;
- tools/compile_grammar_cache.cpp:35;
- tools/grammar_relation_table.cpp:68.

Schema gaps. These are "type": "string" with no `enum`, so `schema_validation` cannot catch a misspelt value:

- labelPlacementH/V in the grammar and scheme schemas;

- placementH/V, labelVAlign, labelHAlign and labelFlush in the landscape schema (landscape.schema.json:587–594).

Recommendation: one descriptor per closed vocabulary. It is an array of {enumerator, wire string} in the header that declares the enum. One generic helper derives the refusing from-string and the to-string from it. Then:

- a loader refuses an unknown value at the boundary (GR-1), with no fallback converters;
- a GUI combo is built from the descriptor and stores the enumerator, never its index. Display texts stay in qtgui, keyed by enumerator, so eacmplib stays Qt-free and translations stay with tr();
- a vocabulary parity test holds each descriptor against its schema enum. That makes the schemas complete and keeps them so.

Adding a value then means one descriptor line, plus the schema and the reference, and the test catches a missing one. This is the *Derived coverage* rule applied to vocabularies. The precedents are scheme_tokens.h with audit rule [token], referral_chains.h with referral_chain_test, and the field registries with field_parity_test.

Open vocabularies stay strings by design: type, shape, colour and scheme names are defined by grammars. The same applies to field *names* (§8), which are a separate question.

7. Magic numbers

Counts: 8,729 numeric literals in production.

- 6,624 are 0, 1 or 2.
- 322 sit in named definitions (constexpr, const, #define).
- 1,783 others appear inline.

The 1,783 include some values that do live in named accessor functions, such as the QColor values in gui_constants.h. The scanner cannot tell those apart.

Pattern	Evidence
N1. The named constant exists, but the literal is used	TOP_BAR_BUTTON_SIZE and WELL_CELL_SIZE are 22 (gui_constants.h:20, :21), yet the literal 22 appears 26 times in view_window.cpp (setFixedSize(22, 22) and similar), which uses the name once
N2. Colour channel bound	255 appears 79 times in 16 files (view_gui_rendering.cpp 30), with 126 literals inside QColor(...). EACMP_Color's 0–255 range is stated in a comment (types.h:192), not by a constant
N3. Tolerances	1e-6 (35×, 9 files), 1e-9 (24×, 9 files) and 0.001 (27×, 7 files) for comparable geometric questions; no named tolerance
N4. Z-order	55 literals in setZValue: zVal + 0.4, 0.41, 0.39, 0.385, 0.393 + 0.001·i, 0.405 + 0.001·i (view_gui_rendering.cpp:227–301), and 1e8 (view_gui.cpp:926). The stacking order of a shape's parts is encoded in decimals across files
N5. Numbers that stand for enumerators	stackAxis 0/1/–1; log events 1/2/4/7; combo index ±1 for the enum position (§3)
N6. Qt layout metrics	setSpacing(8) 41×, setContentsMargins(4...)/(6...), setFixedWidth(60/80/90), resize(400...820), spin ranges 500/10000 (makeLabelAreaSpin 12×)
N7. Algorithm parameters	layout_ogdf.cpp parameter ranges (10000, 1000, 0.1)
N8. Lookup tables	script_input_mac.mm:31–37 (macOS key codes; Carbon names them kVK_ANSI_A and so on), uuid.cpp (bit layout), tools/build_mt_fixtures.cpp (fixture coordinates)

Acceptable as they are: halving to reach a centre (0.5, / 2.0; 85 inline 0.5), and angle arithmetic over 90/180/360 where the expression itself says what it is.

Recommendation. Use inline constexpr, not #define: it is typed, scoped and debuggable, and the code already has 420 constexpr against 4 numeric #defines. The constants belong with their subject:

- eacmplib domain values in their own header, as EACMP_LABEL_AREA_GAP_PT in label_geometry.h;
- GUI metrics in gui_constants.h;
- one tolerance header;
- z-layers as one ordered table;

- EACMP_COLOR_CHANNEL_MAX;
- the numbers that stand for enumerators replaced by enums (§6).

8. Magic strings

Counts: 14,245 string literals in production.

- 1,261 are `tr()` texts.
- 1,588 are messages.
- 1,707 are JSON key reads (539 distinct keys).
- 597 are compared with `==/!=` (340 distinct values).
- 290 are returned.
- The rest are mixed, including JSON keys in writers’ initialiser lists, which the scanner does not classify.
- **JSON keys.** 141 keys are read in two or more files. A shared word does not always mean one key. `autoLayout`, for example, is a grammar section, a scheme key, a model key, a view entry key and a preference block, which are separate formats. The real drift risk is a key spelled again inside one codec. In `model.cpp:1184–1196` the field registry names `"autoLayout"`, and its codec spells it four more times. Recommendation:
 - codecs take the key from their field descriptor;
 - each format’s key lists (accepted sections, carried sections such as `bridge_model_grammar.cpp:435` and `grammar_load.cpp:830`) come from one table per format.
- **Protocol tokens compared as text.**
 - the wildcard `"*"`: 15 compares in 4 files;
 - the `"pcre:"` prefix (`grammar_load.cpp`);
 - `"true"/"false"` as text;
 - the grammar file suffix `".eacmpgrammar.json"`: `app_manager.cpp:1043`, `grammar_load.cpp:31`, `bridge_model_grammar` and four tools;
 - the script locator kinds in `script_sight.cpp`: `"dialog"`, `"row"`, `"menubar"`, `"well"`, `"menu"`, `"browser"`, `"dock"`, `"canvas"`, `"window"`.

Recommendation: named constants, enforced by extending the `[token]` rule from `"@@"` to a listed set of protocol tokens. Locator kinds are a closed vocabulary, so they get a descriptor (§6).

- **A display text used as a sentinel.** `"(default)"` is put into value lists and compared later (`view_window.cpp:3381`, `:4824`, `:4904`), while the combos show `tr("(default)")`. Text must not carry state. The “follows the default” state should be explicit: an optional value or an enumerator.
- **Empty means something.** 43 comments say `“(empty = unset)”` or similar. For example `grammar_load_schemes.cpp:98–99`, and `grammar_load.cpp:791`, where an empty action together with `isDerived` selects a behaviour. That second site is also backward-compatibility code, which the project’s rules exclude. `~/claude/CLAUDE.md` asks for explicit values. Recommendation: the JSON read layer returns an optional for an absent field, so emptiness never stands for “not given”.

9. Keeping track: audit and build

Planned changes to `tools/audit_comments.py`, each documented in `EACMP-documentationincode.md §9`:

Rule	Change
<code>[else]</code> (rule 5)	Close H1–H3. A chain without a final <code>else</code> needs the marker in the <i>last</i> branch’s opening comment. Count a jump-terminated branch as complete, pending decision D1.
<code>[flow]</code> (rule 2)	Cover <code>} else if</code> and one-line bodies (H2, H3).
<code>[switch]</code> (new)	(a) A switch with no default: carries an exhaustiveness marker in its opening comment, the counterpart of no <code>'else'</code> (D2). (b) A default: in a switch whose labels are all of one project enum is flagged (S4). (c) After a no-default switch whose cases all return, the next statement must be <code>EACMP_BREAKING_RECOVERABLE_INVARIANT</code> (S2).
<code>[vocab]</code> (new)	A string literal from a schema enum set compared with <code>==/!=</code> outside a vocabulary descriptor. The sets are read from <code>schemas/*.json</code> , so the list lives in one place.

Rule	Change
[number] (new)	A numeric literal outside a named definition, with 0, 1, 2 and halving exempt. An accept list works like name_audit_accept.txt, for lookup tables and specification constants.
[token] (rule 7)	Extended to the listed protocol tokens (§8).

- **Ratchet.** The repaired and new rules start with thousands of findings. A committed baseline of counts per file and per rule lets the audit report every file whose count grew. Old debt can then be worked off while new code adds none. Today the audit is advisory, so the owner decides whether growth blocks (D4).
- **Build.**
 - Add `-Werror=switch` (clang/gcc) and `/we4062` (MSVC) to `eacmp_compile_contract` (CMakeLists.txt:33–39). Only switch warnings become errors, so the four existing warnings (S1) do not block this. EACMP-build-and-deployments.md follows.
- **Compiler survey.** An on-demand, macOS-only target that repeats this document's syntax-only pass over the compilation database, with `-Wswitch-enum` and `-Wcovered-switch-default`. It reports what the scanner cannot judge (exhaustiveness, dead defaults) without making those flags part of every build. That matters because `-Wswitch-enum` also fires on the four legitimate third-party switches.
- **Tests.**
 - a vocabulary parity test: descriptors against schema `enums`;
 - load tests that refuse an unknown placement and a half pair, in a grammar and in a landscape;
 - a test that a misspelt rule action is refused.

10. Order of work

Each phase is its own commit. Measure first, so that every later phase shrinks a number the audit reports.

1. **Audit and build.** Repair `[else]/[flow]`; add `[switch]`, `[vocab]`, `[number]` and the ratchet baseline; add `-Werror=switch` and `/we4062`; add the survey target. Update §7.1/§9 and the build document.
2. **Silent acceptance (F2).** Rule action and `hideRelations` refuse unknown values. This is a bug fix with a logbook row.
3. **Vocabulary descriptors (§6).** Add the generic helper. Convert classes B to E to refusing descriptors, completing the schemas, with the parity test. Collapse the log-level parsers into one.
4. **Label placement (§3).** Typed per-axis properties, the resolved type without a half pair, loader and setter refusals, schema `enum`, combos from the descriptor. Check the stock landscapes, the corpus and the fixtures for half pairs, and repair any through the migration tool.
5. **Switches (§4).** S2 trailing invariants, S4 explicit enumerators, S5 dead defaults, S6 checked int-to-enum conversions (Qt item data, combos, compiled cache), S7 unscoped enums.
6. **if chains (§5),** by area: loaders, import and serialization first, then the engines, then qtgui, then `mechanical_tests`.
7. **Literals (§7, §8),** by theme: z-layers, tolerances, colour bounds, GUI metrics (N1 first), protocol tokens, the "(default)" sentinel, optional reads in place of empty strings.

11. Decisions for the owner

#	Question	Recommendation
D1	Does a jump-terminated branch count as having its <code>else</code> ?	Yes; the marker only where the false case falls through.
D2	Does an exhaustive switch with no default: carry a marker comment?	Yes, short (for example <code>// exhaustive: -Wswitch</code>), checked by <code>[switch]</code> .
D3	Is <code>-Wswitch-enum</code> a build flag or an on-demand survey?	A survey, plus <code>[switch]</code> (b) in the audit. <code>-Werror=switch</code> in the build now.
D4	Does audit growth block, or stay advisory?	Growth of <code>[else]</code> and <code>[switch]</code> blocks in <code>check-full</code> ; the rest stays advisory.
D5	Label placement's resolved type: <code>std::variant</code> of <i>shape area</i> and <i>box (H, V)</i> , or a mode field beside the box values?	The variant: a mode field would leave H and V present but meaningless in shape mode.

#	Question	Recommendation
D6	Where do vocabulary display texts live?	In qtgui, keyed by enumerator; the descriptor holds only the enumerator and the wire string.